OpenFlow MPLS and the Open Source Label Switched Router

James Kempf, Scott Whyte, Jonathan Ellithorpe, Peyman Kazemian,
Mart Haitjema, Neda Beheshti, Stephen Stuart, Howard Green

james.kempf@ericsson.com, swhyte@google.com, jdellit@stanford.edu,
peyman.kazemian@stanford.edu, mah5@cec.wustl.edu, neda.beheshti@ericsson.com,
sstuart@google.com, howard.green@ericsson.com

ABSTRACT

Multiprotocol Label Switching (MPLS) [3] is a protocol widely used in commercial operator networks to forward packets by matching link-specific labels in the packet header to outgoing links rather than through standard IP longest prefix matching. However, in existing networks, MPLS is implemented by full IP routers, since the MPLS control plane protocols such as LDP [8] utilize IP routing to set up the label switched paths, even though the MPLS data plane does not require IP routing. OpenFlow 1.0 is an interface for controlling a routing or switching box by inserting flow specifications into the box's flow table [1]. While OpenFlow 1.0 does not support MPLS[1], MPLS label-based forwarding seems conceptually a good match with OpenFlow's flow-based routing paradigm. In this paper we describe the design and implementation of an experimental extension of OpenFlow 1.0 to support MPLS. The extension allows an OpenFlow switch without IP routing capability to forward MPLS on the data plane. We also describe the implementation of a prototype open source MPLS label switched router, based on the NetFPGA hardware platform [4], utilizing OpenFlow MPLS. The prototype is capable of forwarding data plane packets at line speed without IP forwarding, though IP forwarding is still used on the control plane. We provide some performance measurements comparing the prototype to software routers. The measurements indicate that the prototype is an appropriate tool for achieving line speed forwarding in testbeds and other experimental networks where flexibility is a key attribute, as a substitute for software routers.

Categories and Subject Descriptors

C.2.6 [**Computer-Communication Networks**]: Internet-working—*Routers*; C.2.2 [**Computer-Communication Networks**]: Network Protocols; C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design; C.2.5 [**Computer-Communication Networks**]: Local and Wide-Area Networks; C.2.6 [**Computer-Communication Networks**]: Internetworking.

General Terms

Design, Experimentation, Management, Performance.

Keywords

OpenFlow, MPLS, NetFPGA, open source LSR.

[1] The latest version of OpenFlow, OpenFlow 1.1, does contain support for MPLS.

1. INTRODUCTION

The OpenFlow 1.0 control protocol [1] provides a vendor agnostic flow-based routing interface for controlling network forwarding elements. The essence of OpenFlow is the separation of the control plane and data plane in routing and switching gear. In traditional routers and switches, the control and data plane are tightly intertwined, limiting the implementation and deployment options. Networks deployed with traditional routing and switching gear have a distributed control plane, and the control and data plane hardware and software for the routing and switching gear is contained in a single box. The OpenFlow interface simplifies the control plane on network forwarding hardware in which the control and data plane are bundled by providing a standardized interface between the control and data planes, simplifying the interface between the on-box control and data plane software and hardware. Alternatively, the control plane can be deployed on a centralized controller that controls multiple forwarding elements, or it can be deployed on a single forwarding element like traditional routers and switches, but with OpenFlow acting as a the control to data plane interface.

In this paper, we describe an extension of OpenFlow to incorporate MPLS and its use in implementing a open source MPLS label switched router (LSR). As far as we are aware, this is the first implementation of MPLS in OpenFlow 1.0. After a brief review of the OpenFlow 1.0 architecture, we describe a modification to the OpenFlow switch data plane model - the virtual port - which supports MPLS encapsulation and decapsulation. We briefly describe the design of OpenFlow MPLS, and the hardware implementation, in NetFPGA [4]. Extensions to OpenVSwitch [6], and the Stanford user space and Linux kernel space OpenFlow reference software switch were also implemented but are not described here. An initial report of this work appeared at MPLS 2010 [7]. We then describe the control plane for the open source LSR that was constructed using the Linux Quagga MPLS distribution [5]. The MPLS Label Distribution Protocol (LDP) [8] is used as a control plane for distributing label switched paths in a network consisting of standard IP/MPLS routers and a collection of OpenFlow MPLS NetFPGA devices configured as LSRs. The on-box OpenFlow controller programs the NetFPGA hardware using the labels distributed by LDP. Unlike standard IP/MPLS networks, the NetFPGA LSRs only utilize IP forwarding on the control plane, to allow communication between the controller and the LSRs. All data plane forwarding is done with MPLS. We provide performance measurements comparing the open source LSR switching performance with Quagga Linux software MPLS forwarding performance. We then conclude the paper with some remarks about the future potential of OpenFlow MPLS.

This paper was peer reviewed by subject matter experts for publication in the Proceedings of ITC 2011

2. OpenFlow MPLS Architecture

Since OpenFlow MPLS is built on top of OpenFlow 1.0, we briefly review the OpenFlow architecture here, and compare it with previous control/data plane separation work before describing the OpenFlow MPLS architecture.

2.1 OpenFlow Architecture

In the canonical OpenFlow 1.0 architecture, the control plane in network switching and routing equipment is moved into a separate controller. The controller communicates over a secure channel with the switches through the OpenFlow protocol. Software running on the controller "programs" the switches with flow specifications that control the routes of packets through the network. For routing purposes, the switches only need to run an OpenFlow control plane, considerably simplifying their implementation. An alternative architecture, shown in Fig. 1, utilizes OpenFlow as the interface between the control and data planes in the same box, while the control plane talks standard IP routing protocols with standard network routers and switches. In this architecture, OpenFlow's flow-based routing design simplifies the on-box control plane/data plane interface. The open source LSR is designed according to the latter architecture.

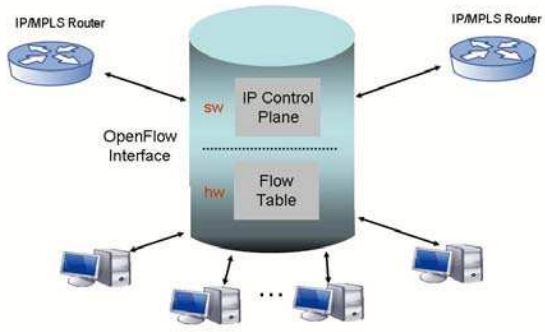

Fig. 1: Single Box OpenFlow Routing Architecture

The switch data plane in OpenFlow is modeled as a flow table in which there are three columns: rules, actions, and counters. The rules column defines the flow. Rules are matched against the headers of incoming packets. If a rule matches, the actions from the action column are applied to the packet and the counters in the counter column are updated. If a packet matches multiple rules, the rule with the highest priority is applied. Each rule consists of elements from a ten-tuple of header fields (see Fig. 2, from [9]) or a wild card ANY. The set of possible actions are: forward as if OpenFlow were not present (usually utilizing the Ethernet spanning tree route), forward to the control plane, forward out a specific port, and modify various header fields (e.g. rewrite MAC address, etc.).

In Port	VLAN ID	Ethernet			IP			Tranport	
		SA	DA	Type	SA	DA	Proto	Src	Dst

Fig. 2: Ten-tuple for Rule Matching

2.2 Previous Work

Much previous work exists in the area of control/data plane separation for routing and switching. Most of the work is specifically directed at implementing the control plane on a separate box from the data plane.

A standard for the separation of the control and data plane in circuit-switched networks is defined by RFC 3292 in the Generalized Switch Management Protocol (GSMP) [2]. GSMP models the network elements as cross-bar circuit switches. Particularly in optical circuit-switched networks, the switches often have physically separate control networks since it is often not possible to intercept the control packets from the optical lambdas, so separation of control and data plane becomes a necessity, GSMP provides a standardized protocol for those systems. Many vendors also have proprietary control protocols.

The FORCES protocol [10] provides a standard framework for controlling data plane elements from a separate controller, but unlike OpenFlow, FORCES does not define a protocol for controlling a specific forwarding element. The FORCES forwarding element model [11] is quite general. OpenFlow, in contrast, defines a specific forwarding element model and protocol involving a flow table and ports. FORCES requires each logical forwarding block in the forwarding element to define its own control protocol within the FORCES framework.

The OpenFlow architecture is perhaps closest to the Clean Slate 4D architecture defined by Greenberg, et. al. [12] The 4D architecture re-factors the network control, data, and management planes into 4 new planes (the "D"s in the name): the decision plane, the dissemination plane, the discovery plane, and the data plane. The data plane is much as before, the decision plane is the OpenFlow controller software, for example NOX [15], and the dissemination plane is provided by the OpenFlow protocol. The OpenFlow architecture defines no special support for discovery. In deployed OpenFlow systems, the controller provides this function through legacy protocols such as LLDP [18]. The Tesseract system [13] implements the 4D architecture.

GMPLS [19] provides another architectural approach to control/data plane separation by extending MPLS to networks including circuit switches. GMPLS utilizes extensions of intradomain routing protocols to perform topology discovery, and RSVP and LMP to establish label-switched paths between network elements. A network element under GMPLS control can either also perform forwarding, in which case GMPLS acts as the control plane for a standard switch or a router, or the network element can control separate forwarding elements through a different forwarding element control protocol. If the latter, a separate switch control protocol, such as GSMP, controls the switches. GMPLS is restricted to transport networks, it does not provide support for IP routing even though it uses IP intradomain routing protocols for connectivity discovery.

There has also been work on control/data plane interfaces for conventional router implementations when the control and data plane are implemented on the same box. The Click modular router toolkit [14] defines interfaces between data plane components that allow modular data planes to be built, but Click does not specify any control plane interface. The control plane interface is hidden behind the individual Click elements. The Xorp router platform [15] defines a composable framework of router control plane processes, each of which is itself composed of modular processing stages. Xorp defines an interface to the data plane through the Forwarding Engine Abstraction (FEA). The FEA interface allows different types of data plane implementations, for example Click or the NetBSD data plane, to be coupled to the control plane without having to change the entire control plane software base. The Conman architecture [16] defines a modular data plane and control plane architecture with a well defined pipe interface between the two. Our work differs from prior work in this area in that we have taken an interface that was defined for simplifying and centralizing the control plane and instead implemented it as the control/data plane interface on a single box, providing a more flexible deployment model for cases where a centralized control plane is impractical.

2.3 OpenFlow MPLS

2.3.1 OpenFlow MPLS Design

MPLS forms flow aggregations by modifying the packet header to include a label. The label identifies the packet as a member of a forwarding equivalence class (FEC). A FEC is an aggregated group of flows that all receive the same forwarding treatment.

A data plane MPLS node implements three header modification operations:

- *Push*: Push a new label onto the MPLS label stack, or, if there is no stack currently, insert a label to form a new stack,
- *Pop*: Pop the top label off the MPLS label stack,
- *Swap*: Swap the top label on the stack for a new label.

In Port	VLAN ID	Ethernet			MPLS		IP			Tranport	
		SA	DA	Type	Label 2	Label 1	SA	DA	Proto	Src	Dst

Fig. 3: OpenFlow Twelve-tuple for MPLS rules

The MPLS label stack is inserted between the IP and MAC (Layer 3 and Layer 2) headers in the packet. MPLS label stack entries consist of 32 bits, 20 of which form the actual label used in forwarding. The other bits indicate QoS treatment, top of stack, and time to live.

The first modification required to OpenFlow is to increase the size of the tuple used for flow identification. In principle, the size of the MPLS label stack has no upper bound, but as a practical matter, most carrier transport networks use a maximum of two labels: one label defining a service (such as VPN) and one label defining a transport tunnel. We therefore decided to extend the header tuple used for flow matching from 10 fields to 12. Only the actual 20 bit forwarding label is matched, the other bits are not included. Fig. 3 shows the 12 tuple.

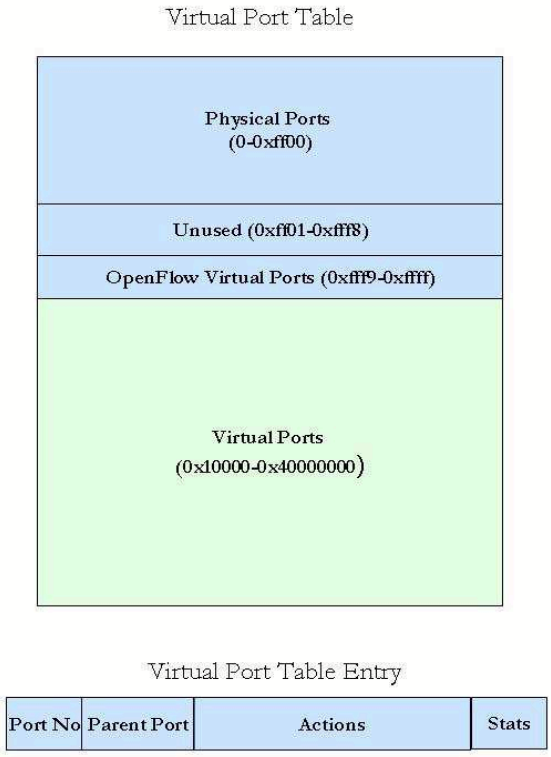

Fig. 4: Virtual Port Table and Virtual Port Table Entry

The next required modification was the addition of the MPLS header modification actions (push, pop, and swap) to the action set executed when a rule matches. With the exception of limited field rewriting, OpenFlow 1.0 actions perform simple forwarding. The MPLS push and pop actions, in contrast, rewrite the header by inserting fields into the header. Rather than inserting the MPLS protocol actions into the basic OpenFlow packet processing pipeline, we chose instead to isolate them using a abstraction called a *virtual port*. A virtual port is an abstraction mechanism that handles complex protocol specific actions requiring header manipulation, thereby hiding the complexity of the implementation. This allows yet more complex header manipulations to be implemented by composing them out of simpler virtual port building blocks.

Virtual ports can be hierarchically stacked to form processing chains on either input or output. On output, virtual ports can be included in flow table actions just like physical ports. Virtual ports are grouped together with physical ports into a virtual port table. Fig. 4 illustrates the virtual port table, together with a table row. Each virtual port table row contains entries for the port number, the parent port, the actions to be performed by the virtual port, and statistics.

The MPLS actions in the virtual port table consist of the following:

- *push_mpls*: Push a 32 bit label on the top of the MPLS label stack , and copy the TTL and QoS bits from the IP header or previous MPLS label,

- *pop_mpls*: Pop the top label on the MPLS stack, and copy the TTL and QoS bits to the IP header or previous MPLS label,
- *swap_mpls*: Swap the 20 bit forwarding label on top of the MPLS stack.
- *decrement_ttl*: Decrement the TTL and drop the packet if it has expired.
- *copy_bits*: Copy the TTL and QoS bits to/from the IP header or previous MPLS label

We also added a counter to the OpenFlow statistics that is incremented every time a virtual port drops a packet due to the expiration of the TTL.

The OpenFlow protocol was extended with the following messages to allow the controller to program label switched paths (LSPs) into the switches:

- *vport_mod*: Add or remove a virtual port number. Parameters are the parent port number, the virtual port number, and an array of virtual port actions,
- *vport_table_stats*: Return statistics for the virtual port table. The statistics include maximum virtual ports supported by the switch, number of virtual ports in use, and the lookup count, port match count, and chain match count.
- *port_stats*: The OpenFlow *port_stats* message applies to virtual ports as well, but only the *tx_bytes* and *tx_packets* fields are used.

Finally, the OpenFlow *switch_features_reply* message was modified to include a bit indicating whether the switch supports virtual ports.

2.3.2 NetFPGA Implementation

NetFPGA is a PCI card that contains a Virtex-2 Xilinx FPGA, 4 Gigabit Ethernet ports, SRAM and DDR2 DRAM [4]. The board allows researchers and students to build working prototypes of line speed network hardware.

The MPLS implementation extends the OpenFlow 10 tuple with two additional fields for MPLS labels, and adds virtual port functionality to support MPLS-specific actions. Fig. 5 shows the functional block diagram of the NetFPGA design. Our implementation of OpenFlow MPLS in NetFPGA is based on the OpenFlow v0.89 reference implementation for NetFPGA. OpenFlow v0.89 differs slightly from OpenFlow 1.0 in that OpenFlow 1.0 supports VLAN type and IP ToS headers whereas v0.89 doesn't. We used v0.89 because it was available at the time the work was done, since these features aren't necessary for the open source LSR and would have taken up valuable FPGA memory (only 5% of NetFPGA Virtex-2 FPGA remained empty after implementing MPLS OpenFlow), we decided not to update.

As packets arrive, a lookup key is created by concatenating the 12 fields together. The lookup key is used in parallel by two lookup engines, one performing exact match using two CRC hash functions and the other one doing wildcard match using a TCAM. Each of the exact and wildcard tables has 32 entries. The result of these lookup operations is fed into a match arbiter that always prefers an exact match to a wildcard match. The OpenFlow actions associated with the match are then performed. If the action involves a port, the port number is checked to see if the number matches a virtual port. If it does, the virtual port header manipulation actions are performed.

In the OpenFlow MPLS implementation, virtual ports implement the MPLS actions: push a new label, pop the top of the stack label, decrement the TTL, copy the TTL and copy the QoS bits. As an optimization, the swap operation is handled by an OpenFlow *rewrite* action instead of in the virtual port. If the *copy_bits* action is performed during a push operation, it copies the TTL/QoS bits from previous MPLS label, and if it is done as part of pop operation, the TTL/QoS bits of current label are copied to the previous label. If only one MPLS label exists, IP TTL or IP ToS is the source or target instead. The *decrement_ttl* action decrements the TTL value for the top of the stack label and drops the packet when the label value hits zero. To decrement the MPLS TTL, without any push/pop operation or as part of a swap action, the packet is forwarded to a pop virtual port with the pop and copy TTL/QoS functionality disabled.

Virtual ports can be concatenated together for up to two layers to perform two push or two pop operations in one NetFPGA card. The last virtual port in the chain forwards the packet to a physical port on output, or the first virtual port accepts a packet from a physical port on input.

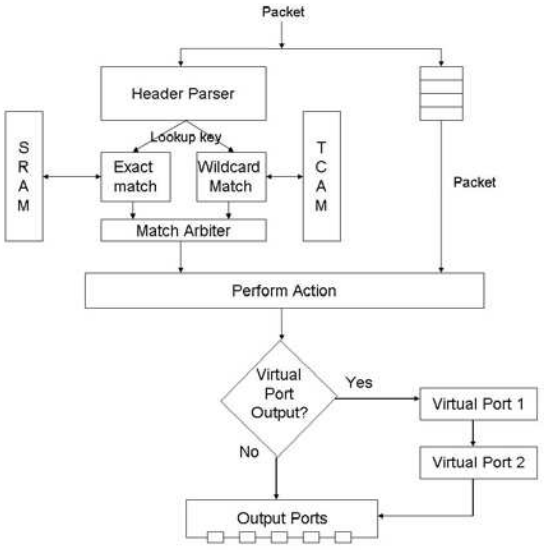

Fig. 5: OpenFlow-MPLS on NetFPGA Block Diagram

The last 8 positions in the wildcard table are always filled by default entries to handle forwarding unmatched packets to the control plane. For each of the 4 NetFPGA ports, there is one entry at the bottom of the wildcard table that has everything except an incoming port wildcarded. If a packet doesn't match any other entry in the table, it will at least match that default entry and is forwarded to the DMA port corresponding to its input port. The packet is then received by the OpenFlow kernel module running on the host machine and is forwarded to the control plane. Similarly, packets coming from the control plane, are sent out on a DMA port by the OpenFlow kernel module, and are received by NetFPGA. There are 4 default rules in the wildcard table that match on the packets coming from each of the 4 DMA ports and forward them to corresponding output port.

3. Open Source Label Switched Router

3.1 OpenFlow-MPLS LDP Control Plane

As a demonstration of OpenFlow MPLS, we built a low-cost label switched router (LSR) consisting of a NetFPGA board running OpenFlow MPLS in a PC running the Linux OpenFlow driver, and the Quagga open source routing stack, including Quagga LDP and MPLS Linux [22]. In this model, the OpenFlow controller runs on the same box as the NetFPGA LSR and acts as the control plane only for the NetFPGA on the box as is the case in standard routers and switches, in contrast to the canonical OpenFlow model discussed in Section 2.1. Fig. 6 contains a block diagram of the open source LSR. The entire bill of materials for the open source LSR was around $2000.

LDP, the Label Distribution Protocol [8], connects the open source LSR with other forwarding elements in the network. LDP allows two devices to form an adjacency and establish label bindings for label switched paths between them. An LDP neighbor sends a LDP packet to the open source LSR in-band on one of its connected interfaces. The open source LSR identifies the packet as part of a LDP flow and forwards it to the control plane on the box, where it is sent to the Quagga LDP daemon. As is the case for IP-MPLS routers, the open source LSR exchanges OSPF route information with external routers so that MPLS paths can be established along known IP routes.

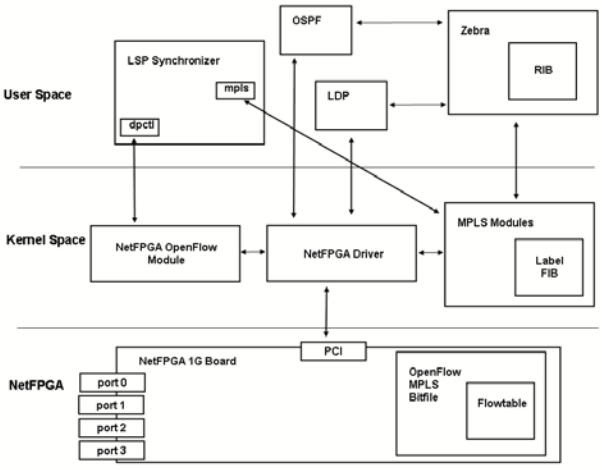

Fig. 6: Open source LSR Block Diagram

The LDP Daemon maintains and builds a normal LDP adjacency. Once LDP has formed an adjacency and completed a label binding, it updates the kernel MPLS LFIB with the corresponding label information. The LSP Synchronizer is a user level daemon that polls the MPLS LFIB in the kernel periodically for changes, and when it detects a change it pushes a an OpenFlow flow modification into the NetFPGA, enabling data plane packets received with those labels to be forwarded correctly in hardware.

3.2 Performance Measurements and Interoperability Verification

The open source LSR is primarily a tool for prototyping new ideas in networking, in that it offers line speed performance with the flexibility to change control plane and data plane software. Consequently, we performed a couple of simple performance tests against the MPLS Linux to demonstrate the performance advantage. The tests measured bidirectional throughput for packets of 68 byes or 1504 bytes in length on a single port. The results are shown in Fig. 7. As should be clear from the figure, there is 2 orders of magnitude difference in forwarding performance between the NetFPGA and MPLS Linux. In addition, the performance of the NetFPGA LSR was constant regardless of packet size, whereas the performance of MPLS Linux decreased for smaller packets.

The NetFPGA was able to maintain line speed performance up to 3 ports, but scaled down to 6 Giga packets/second at 4 ports. This limitation had nothing to do with the MPLS implementation, other NetFPGA applications exhibit the same performance profile. Note that many carefully coded and highly optimized software routers are able to achieve much better performance than MPLS Linux exhibited in this study, but our objective here is not to compare the best software router with hardware, but rather to show the open source LSR provides good, reasonably scalable performance in comparison with a widely available, off the shelf software implementation.

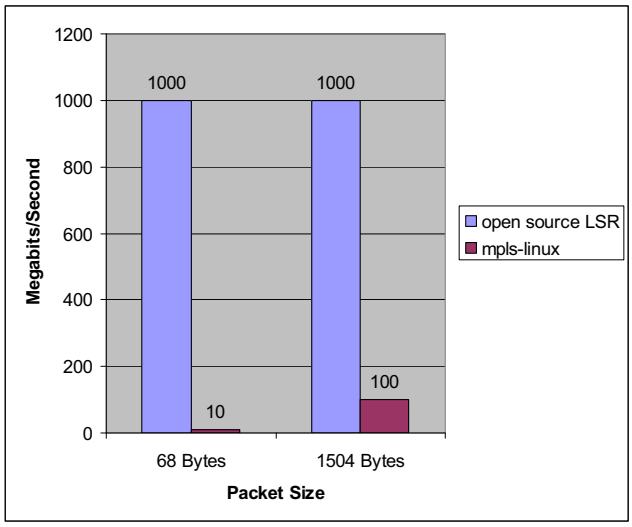

Fig. 7: Comparative Forwarding Performance

We also performed a test to verify that the open source LSR could be used in a network consisting of standard IP/MPLS routing gear running the standard IP/MPLS LDP/OSPF control plane. The network, shown in Fig. 8, consisted of two standard IP/MPLS routers, the SmartEdge100 [20] and the Juniper M10. All devices were running OSPF and LDP. The open source LSR was able to exchange OSPF and LDP messages with the IP/MPLS routers to set up LSPs through the network.

Finally, we set up a test network to verify that it is possible to perform MPLS forwarding within a core network without requiring iBGP. The core test network is shown in Fig. 9. Again, all devices speak OSPF and LDP. Two Juniper M10 routers function as label edge routers (LERs) and speak iBGP. The iBGP packets are routed through the network of open source LSRs. Two hosts serve as sources and destinations of traffic. The LERs push and pop MPLS labels onto/off of the host traffic packets to route through the open source LSR core. Note that while OpenFlow MPLS is designed to allow an OpenFlow MPLS switch to act as an LER too, in this case, we wanted to use the M10s to

demonstrate how the open source LSR could be used to set up a iBGP free core. The addition of a BGP module to the control plane on the open source LSR would allow it to act as an LSR.

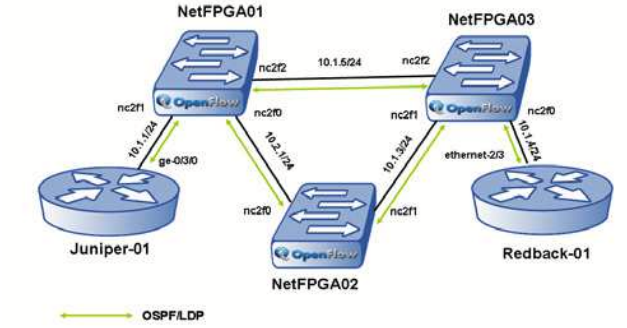

Fig. 8: Interoperability Test Network

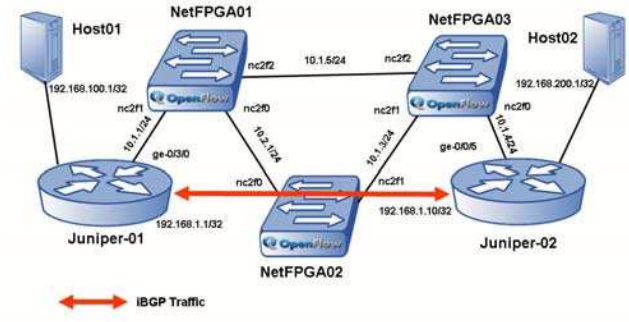

Fig. 9: Core Interoperability Test Network

4. Summary and Conclusions

In this paper, we described an extension of the OpenFlow forwarding element control interface and model to include MPLS. The extension involves modifying the OpenFlow flow table entries to include two MPLS labels and defining an extension of the OpenFlow port model to allow definition of virtual ports. The virtual ports implement the MPLS per packet processing operations. An extension to the OpenFlow protocol allows the controller to program the OpenFlow MPLS forwarding element to match and process MPLS flows.

The extension to OpenFlow was implemented on OpenVSwitch, the Stanford reference software switch (both user and kernel modules), and on the NetFPGA hardware. The NetFPGA implementation supports up to 2 virtual ports for MPLS. A demonstration system, the open source LSR, was built using OpenFlow as the control/data plane interface for a NetFPGA running on the same PC as the control plane. This use of OpenFlow is in contrast to the canonical architecture in which a whole network of switches is controlled by one OpenFlow controller.

Some simple performance tests were run comparing the open source LSR to MPLS Linux for prototyping purposes. The tests demonstrated that the open source LSR could significantly improve the performance of forwarding in prototype networks. An interoperability demonstration system was built using the open source LSR and two standard Internet routers capable of MPLS routing. The routers exchange LDP with the open source LSRs and each other to set up LSPs through the network. Finally, a prototype iBGP free core network was set up that performs MPLS forwarding without the need for IP routing or iBGP speakers. The network consisted of two hosts connected up to routers and a collection of open source LSRs. No interoperability problems were found.

Going forward, the success of MPLS in OpenFlow 1.0 has led to the incorporation of MPLS into the next version of OpenFlow, OpenFlow 1.1. Support for MPLS in the canonical OpenFlow centralized control plane model is necessary to utilize MPLS in OpenFlow 1.1. For the open source LSR model, the current NetFPGA 1G only supports a realistic maximum of 32 flows, which is really too few for production use, even in a small campus testbed. The NetFPGA 10G [21] is a much better platform and is the target for future work. In addition, a port to NetBSD is planned, since the MPLS implementation in NetBSD is more stable. Code for the open source LSR is available at http://code.google.com/p/opensource-lsr.

5. ACKNOWLEDGMENTS

The authors would like to thank Andre Khan for his founding contribution during the initial phases of the design, and Nick McKeown for his helpful direction during the design of the virtual port abstraction.

6. REFERENCES

[1] "OpenFlow: Enabling Innovation in Campus Networks", McKeown, N., et. al., March, 2008, http://www.openflowswitch.org//documents/openflow-wp-latest.pdf.

[2] Doria, A., Hellstrand, F., Sundell, K., Worster, T., "General Switch Management Protocol (GSMP)", RFC 3292, June 2002.

[3] Rosen, E., Viswanathan, A., and Callon, R., "Multiprotocol Label Switching Architecture", RFC 3031, Internet Engineering Task Force, January, 2001.

[4] http://www.netfpga.org.

[5] http://www.quagga.net.

[6] http://openvswitch.org.

[7] http://www.isocore.com/mpls2010/program/abstracts.htm#wed1_5.

[8] Andersson, L., Minei, I., Thomas, B., "LDP Specification", RFC 5036, Internet Engineering Task Force, October, 2007.

[9] OpenFlow Switch Specification V1.0.0, http://www.openflowswitch.org/documents/openflow-spec-v1.0.0.pdf.

[10] Doria, A., Ed., Salim, J., Ed., Haas, R., Ed., Khosravi, H., Ed., and Wang, W., Ed., "Forwarding and Control Element Separation (ForCES) Protocol Specification", RFC 5810, Internet Engineering Task Force, March 2010.

[11] Halpern, J., and Salim, J., "ForCES Forwarding Element Model", RFC 5812, Internet Engineering Task Force, March 2010.

[12] Greenberg, A. et. al., “A Clean Slate 4D Approach to Network Control and Management,” Proceedings of ACM SIGCOMM, 2005.

[13] Yan, H., Maltz, D., Ng, E., Gogineni, H., Zhang, H., and Cai, Z., “Tesseract: A 4D Network Control Plane”, Proceedings of the 4th USENIX Symposium on Networked Systems Design & Implementation, pp. 369–382, March, 2007.

[14] Kohler, E., Morris, R., Chen, B., Jannotti, J., and Kasshoek, F., “The Click Modular Router”, *Operating Systems Review,* **34**(5), pp 217{231, December, 1999.

[15] Handley, M., Hodson, O., and Kohler, E., “XORP goals and architecture” , Proceedings of the ACM SIGCOMM Hot Topics in Networking, 2002.

[16] Ballani, H., and Francis, P., “CONMan: A Step Towards Network Manageability”, Proceedings of the ACM SIGCOMM Workshop on Internet Network Management, September, 2006.

[17] Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., Shenker, S., “NOX: Towards an Operating System for Networks”, Computer Communications Review, July, 2008.

[18] IEEE standard 802.1ab, “802.1ab rev – Station and Media Access Control Connectivity Discovery”, September, 2009.

[19] Farrel, A. ad Bryskin, I., *GMPLS: Architecture and Applications*, Morgan Kaufmann Publishers, Amsterdam, 412pp., 2006.

[20] http://www.ericsson.com/ourportfolio/network-areas/se100?nav=networkareacategory002%7Cfgb_101_504%7Cfgb_101_647.

[21] http://netfpga.org/foswiki/NetFPGA/TenGig/Netfpga10gInitInfoSite.

[22] http://sourceforge.net/apps/mediawiki/mpls-linux/index.php?title=Main_Page.