SDN Shim: Controlling Legacy Devices

Daniel J. Casey, Barry E. Mullins
Department of Electrical & Computer Engineering
Air Force Institute of Technology
Wright-Patterson Air Force Base, OH 45433 USA
{daniel.casey, barry.mullins}@afit.edu

***Abstract*—We propose a design to aid experimentation with SDN architectures using legacy network equipment. We implement a portion of the design on an FPGA and evaluate throughput and latency. Results indicate viability for testbed and research environments, especially with proposed additions to further reduce latency in broadcast- and multicast-heavy traffic.**

***Index Terms*—Ethernet networks, Field programmable gate arrays, Software defined networking.**

I. Introduction

Upgrading an enterprise network to leverage software-defined networking (SDN) can be difficult and expensive. The most widely-used SDN protocol, OpenFlow, has only been available in commercial switches since 2012. Furthermore, the protocol itself is rapidly evolving, with other SDN protocols under active development. These factors add cost and risk to SDN upgrades, which hinder adoption and growth. Moreover, legacy networking equipment may still provide full functionality. As described in [1], "Certainly, rip-and-replace is not a viable strategy for the broad adoption of new networking technologies". Providing cost-effective means to adopt SDN technologies without completely replacing existing infrastructure benefits researchers and end-users alike.

We propose an inexpensive hardware device that usurps flow-level control from a legacy switch. This "shim" layer can provide SDN features on legacy switches to enable pre-purchase testing and cost-effective infrastructure upgrade planning.

Our goals are to determine a viable shim design, where the shim is best deployed and if, despite performance limitations, such a device is practical. The specific performance metrics are throughput and latency of the device while connected to a switch. The design is implemented on a NetFPGA-1G-CML development board which uses a Xilinx Kintex-7 FPGA connected to four 1 Gigabit per second (Gbps) Ethernet PHY chips and ports. The switch is a Cisco Nexus 3048T. It is not a "legacy" switch as it includes support for OpenFlow and other SDN protocols, but was configured to behave as one for our experiments. It is equipped with 48 1 Gbps Ethernet ports and four 10 Gbps small form-factor pluggable transceiver (SFP+) ports.

II. Related Work

There are a variety of proposed techniques for gaining control of legacy network devices in an SDN network. They generally fall into the categories of controller-based legacy control, intermediate devices, and network architecture planning.

An example of controller-based legacy control is OpenDaylight, a relatively new and open-source SDN controller. It includes a service adaptation layer which aims to abstract the southbound protocol details from the higher layers by use of protocol plugins [2]. In addition to OpenFlow, NETCONF, etc., plugins can be created to cover various legacy switches. Depending on the particular target switch and control devices exposed (CLI, SNMP, web, etc.), this could be difficult. A heterogeneous switch environment would further complicate the effort. Even a well-written plugin for a specific device might not provide the required resolution of control to cover certain use cases, making the investment questionable.

An example of an intermediate device is given in [3], where the authors propose a system to translate OpenFlow messages on-the-fly into legacy command directives for non-OpenFlow switches. They test this on three different vendors' switches, using command-line, SNMP, and web service configuration to modify the switch behavior in accordance with the OpenFlow messages from the controller. This approach has its own drawbacks. Again, it would be necessary to customize the method of configuration control for every vendor, and possibly every model or even software version of a switch. Also, as explained in [1]:

> One fundamental restriction of this approach is sacrificing the reactive mode of operation of OpenFlow, which packets without a matching rule are forwarded to the controller via packet-in events.

As the OpenFlow protocol itself must be altered, many network applications could not run unmodified. Finally, they also found their approach requires substantial modifications to the controller, further distancing their solution from a standard OpenFlow deployment.

ClosedFlow is a more recent effort to include legacy switches in an OpenFlow environment. In [4], the authors target 10-year old Cisco switches with the goal of being able to run unmodified SDN applications. They repurpose the layer 3 routing protocols on the switches, specifically OSPF. The obvious drawback to this approach is that it requires the legacy switches to be multilayer switches that support routing protocols. These switches are much more expensive and typically less abundant than layer 2-only switches.

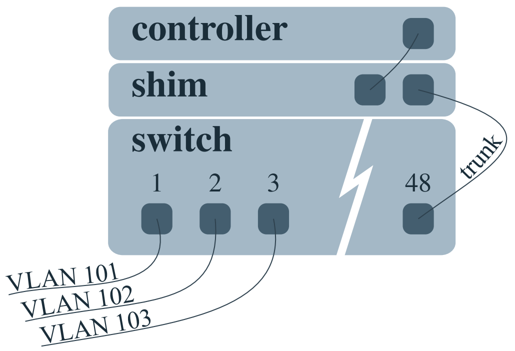

Fig. 1. Physical switch/shim configuration

An example of network architecture planning as an SDN-enabler is given in [5], where the authors propose careful placement of SDN-enabled switches in a network. As long as any flow traversing the network is handled by at least one SDN switch, many of the benefits of SDN can be gained while requiring only a subset of devices be upgraded. Given certain assumptions and a typical, 3-tier enterprise network architecture, they suggest that as few as 10% of the distribution layer switches can be upgraded to achieve SDN management capabilities over the whole network. We believe this approach comes closest to solving the problem, and could be used simultaneously with our proposed shim.

III. Hardware Shim Design

We believe an OpenFlow "shim" layer implemented with relatively inexpensive hardware could be successful in certain scenarios. Our interest is to develop a device that will connect to an OpenFlow controller, present itself to the controller as a regular OpenFlow switch, and control flows on the connected legacy switch in accordance with messages from the controller. Design goals include utilizing ubiquitous switch features, supporting all OpenFlow features, minimizing added latency and reduced throughput, matching the frame rate from the switch, and keeping the hardware design simple (and therefore inexpensive).

The only prerequisite switch feature is support for IEEE 802.1Q virtual local area network (VLAN) tags. This makes the design widely applicable, as VLANs are supported on the vast majority of business class switches manufactured in the last 10 years. An example physical configuration is shown in Figure 1, with a corresponding configuration in Cisco IOS format shown in Figure 2. The legacy switch must be preconfigured with a VLAN trunk to the shim and access ports on unique VLANs. While there are 4094 VLAN IDs available (12 bits with first and last reserved: $2^{12} - 2$), some switches only support a smaller subset of active VLANs. However, this smaller subset is usually more than the number of ports on the device. Switch-connected ports should be configured as trunk ports, but the selection of VLANs for these trunks should be distinct from those used for the host-connected access ports.

As each host port resides on its own VLAN, the switch will never pass frames from one port to another, only to the shim.

```
interface Ethernet1/1
  no lldp transmit
  no cdp enable
  switchport mode access
  switchport access vlan 101
  spanning-tree port type edge
  spanning-tree bpdufilter enable
interface Ethernet1/2
  ...
  switchport access vlan 102
  ...
interface Ethernet1/48
  description netfpga-shim
  ...
  switchport mode trunk
  switchport trunk allowed vlan 101-147
```

Fig. 2. Example switch configuration

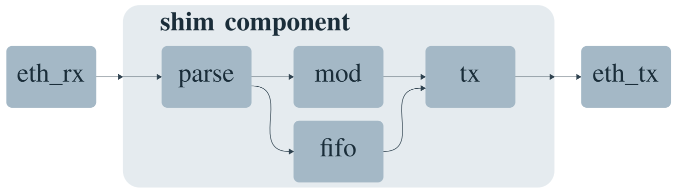

Fig. 3. Design VHDL components

The shim receives all switch traffic, and is able to manage whether each packet is delivered out another port, delivered to the controller via an OpenFlow `PACKET_IN` message, dropped, modified, or rewritten. The shim therefore contains the primitives of a true OpenFlow switch, with the legacy switch providing a physical extension of ports to the FPGA device. This approach should work even on a network where VLANs are already in use, as long as separate VLAN IDs are chosen and the VLANs allowed on each trunk are carefully controlled.

In contrast with other approaches, this design does not require any modification to the SDN controller, and should be able to support all OpenFlow features. It is widely applicable to legacy switches, requiring only VLAN support, and does not have to be customized to the switch make or model. It can be implemented with relatively inexpensive hardware (cost of FPGA board).

IV. Implementation

Our system is written in VHDL with Vivado Design Suite 2014.4, targeting the Xilinx Kintex-7 XC7K325T-1FFG676 FPGA on the Digilent/CML NetFPGA-1G-CML. To keep the design small and fast, we did not use the existing Digilent/CML code base but rather designed our own components. The design is comprised of three top-level components: the receive and transmit interfaces and the shim wrapper. Secondary top-level components include the clock generator, global reset, and PHY reset, along with debugging components (integrated logic analyzer and virtual I/O).

The VHDL components are depicted in Figure 3. The shim wrapper instantiates a shim component for each activated

TABLE I
FIELD-PROGRAMMABLE GATE ARRAY RESOURCE UTILIZATION
(1 PORT, STATIC FLOW, KINTEX-7)

Resource	Utilization	Available	Utilization %
FF	870	407600	0.21
LUT	582	203800	0.29
Memory LUT	7	64000	0.01
BRAM	15	445	0.34

interface (the number of active shim interfaces is configurable). Each shim component includes an Ethernet parser which extracts header details, a modifier which manipulates the header, and an Ethernet generator which combines the modified header with the frame payload into a Gigabit Media Independent Interface (GMII) stream. The modifier references the shim configuration memory to map the incoming frame VLAN to the input port of the external switch, which is analogous to OpenFlow's `uint8_t in_port` portion of a `ofp_packet_in` message. Based on the other header parameters and the configuration memory, the modifier selects whether to transmit a modified version of the frame or drop it. If selected for transmission, the output port is encoded as the new VLAN (again analogous to the OpenFlow `uint16_t out_port`), and the Ethernet generator is signaled to generate the modified GMII stream for transmission. The FIFO queue is used to hold the frame payload.

If the frame must be broadcast or output on multiple ports (`OFP_FLOOD` or `OFP_ALL` output ports in OpenFlow), the modifier instead signals the broadcast generator, which keeps a two-stage FIFO buffer for generating n copies of frame without blocking the primary pipeline. Of course, given enough sequential broadcasts and a smaller parameterized FIFO for the broadcast generator, the system can end up dropping some broadcast frames. However, dropping frames due to contention is allowed in Ethernet.

This system is designed to match the switch's outgoing frame rate. As the shim is a hardware pipeline design, it is able to track one-for-one with frames coming from the switch. Implementing a single port of the design on the NetFPGA-1G-CML uses only a small portion of the FPGA, as shown in Table I (without broadcast offload and using a static VLAN configuration). A custom shim could be developed on a much smaller (and inexpensive) FPGA.

V. MEASUREMENTS AND RESULTS

Throughput was measured with iperf3 [6] and latency was measured with a separate FPGA. The switch was a Cisco Nexus 3048 (N3K-C3048TP-1GE). Switch features that may interfere (e.g., STP, CDP, and LLDP) were disabled for all tests. The test server was an Aberdeen Superserver with 384 GB RAM, two 2.30 GHz 8-core Intel Xeon E5-4610 CPUs, and two 4-port Intel 82576 1 Gbps Ethernet cards running Fedora 21. Pairs of containers were used to perform each throughput test with iperf3, with one container attached to either end of the switch or shim.

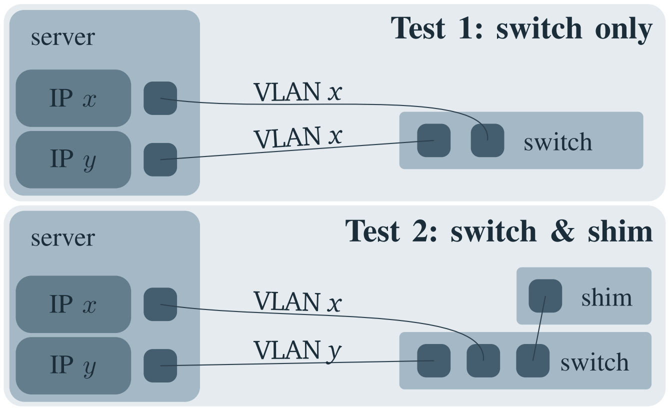

Fig. 4. Overview of test configurations

TABLE II
PERFORMANCE RESULTS

	Avg Throughput	Avg Latency
Switch	943 Mbits/sec	3894.6 ns
Switch & Shim	936 Mbits/sec	7567.6 ns

The throughput test configurations are depicted in Figure 4; latency tests were analogous, but with an FPGA instead of the server for higher resolution timing. Test 1 provides a baseline and measures switch performance, with two switch ports configured as access ports of the same VLAN. Test 2 connects the shim device to the switch as a VLAN trunk, and the two access ports are set to separate VLANs. Table II summarizes the results of each test. The throughput test uses iperf3 with default settings, which attempts to simulate typical application performance by sending a series of 10 TCP streams, each 1 second, and measuring bytes successfully sent. The latency test uses an FPGA that sends a UDP packet once per second over a period of 40 seconds and measures the time until the packet is received to estimate total round trip time. Our latency measurements meet our expectations based on the switch specifications, which indicate latency between 2.7 and 7.2 microseconds. When dealing with single flows, the shim hardly limits network performance. While the increased latency and reduced throughput may be negligible for this simple set up, we expect it to be a larger issue with contention in the switch when there are multiple simultaneous streams that all need to be fed through the shim.

VI. POTENTIAL ISSUES AND MITIGATIONS

The most significant issue is decreased throughput as the configuration creates a bottleneck at the switch port leading to the shim. A single switch port simply cannot handle the traffic of all the other ports at the same time. While this design would not be the best fit for a high-traffic, throughput-sensitive network, there are a few approaches that can help alleviate the issue.

Order-of-magnitude higher-bandwidth ports can be used, if available, on both the shim device and switch. For example, many switches have a few higher-speed uplink ports, e.g., 10 Gbps ports on an otherwise 1 Gbps switch, or 40 Gbps ports on a 10 Gbps switch. A variety of 10 Gbps SFP+ FPGA boards are available at reasonable cost. As another approach, many FPGA development boards have two or four network ports; the bandwidth of these could be combined with a link aggregation protocol, albeit to the detriment of the number of available switch ports.

The design also introduces added latency to each flow. The amount of latency is proportional to the amount of traffic and number of ports the shim is required to handle. The mitigating approaches outlined above also help reduce added latency. Additionally, the choice of FPGA as a platform and the specific implementation of that design keep added latency to a minimum, as opposed to a purely software-based design.

While the design introduces some additional latency, it is less than 10 microseconds for most flows. However, the added latency is more profound with broadcast or multicast traffic. Since each port is on a separate VLAN, there is no shared broadcast domain. As a result, when the shim must deliver a frame to multiple ports, it must sequentially produce a copy of that frame for each destination port. The amount of time required to transmit one broadcast frame will be more than n times that of a traditional switch, where n is the number of ports. This also results in skew between the broadcast times of the first and last ports, which could negatively affect some applications. The time the shim spends sending all these frames precludes it from sending any other traffic, which could result in a backlog of traffic and potentially many dropped frames.

One technique to avoid the broadcast latency issue is to use a feature that retains isolation between host ports while providing a shared broadcast domain, like protected or isolated ports or private virtual LANs (PVLANs). These features are generally the same, but have different names and nuances across vendors and models. Since the shim is on an unprotected or promiscuous port of a primary VLAN, it could send broadcasts to all other ports with a single frame. The issue with using PVLANs is that it removes the ability of the shim to distinguish between individual ports. The shim could potentially target certain ports by keeping track of MAC addresses and relying on the MAC learning of the switch, but the device could no longer be a drop-in replacement supporting OpenFlow.

Another technique to mitigate broadcast latency, resulting backlog, and dropped frames is to utilize a separate port and processing pipeline of the shim for sending broadcasts. This keeps the primary port of the shim dedicated to unicast transmissions only, which are higher priority in most situations. When a broadcast is needed, it is sent to a separate pipeline within the FPGA that transmits packets out a secondary port connected to the switch. The delay for the primary pipeline is identical to that of a unicast frame.

While these approaches can mitigate the issue, the design will always result in some reduction in bandwidth. Whether this is acceptable depends upon the nature of the network; the shim may not be an acceptable choice for a network operating near its maximum capacity. On the other hand, an existing network could be safely partitioned to allow experimental deployment of the SDN architecture on the existing hardware.

VII. Future Work and Conclusion

We plan to continue design development to make it more relevant to high-traffic environments. The first step is to move development to a 10 Gbps board and use 10 Gbps switch uplink ports for the shim trunk. The next step is hand-off between output ports on the shim; the switch will be partitioned into three or four sets of ports, with each set being handled by a different port on the shim. Frames that cross partitions will be transparently handed off in the FPGA; we expect this will improve contention by a factor of the number of partitions. Finally, updating the design to include an OpenFlow agent will make the shim a stand-alone, drop-in SDN-enabler. We will compare implementing the agent as a soft-core processor (MicroBlaze on Xilinx FPGAs) against an ARM-based mezzanine card.

The upgrade path to a modern SDN network architecture can be daunting. Equipment costs are formidable, benefits may be unclear, and networking programming is unfamiliar territory for many network engineers. A simple shim device that can add a layer of SDN functionality to legacy equipment may go a long way toward easing adoption pains for network practitioners. Certainly, there will be trade-offs in such a design in terms of performance. However, these compromises may be acceptable in testbeds as the shim can help SDN newcomers understand how they might leverage new networking technologies, perform more controlled upgrades of their equipment, and get the most long-term value for their investment.

Acknowledgment

The views expressed in this article are those of the authors and do not reflect official policy of the United States Air Force, Department of Defense, or the U.S. Government.

References

[1] F. Hu, *Network Innovation Through OpenFlow and SDN: Principles and Design*. CRC Press, 2014.

[2] J. Medved, R. Varga, A. Tkacik, and K. Gray, "OpenDaylight: Towards a Model-Driven SDN Controller architecture," in *IEEE 15th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2014, pp. 1–6.

[3] F. Farias, J. Salvatti, P. Victor, and A. Abelem, "Integrating Legacy Forwarding Environment to OpenFlow/SDN Control Plane," in *15th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, September 2013, pp. 1–3.

[4] R. Hand and E. Keller, "Closedflow: Openflow-like control over proprietary devices," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking (HotSDN '14)*. New York, NY, USA: ACM, August 2014, pp. 7–12.

[5] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks," in *2014 USENIX Annual Technical Conference (USENIX ATC 14)*. Philadelphia, PA: USENIX Association, June 2014, pp. 333–345.

[6] ESnet & Lawrence Berkeley National Laboratory. iperf3. [Online]. Available: http://software.es.net/iperf/